PROJECT 3 REPPORT

Owner: Marius THORRE

Master 2 IAAA

ABSTRACT

This report explores the implementation of a neural network-based approach for word embedding and language modeling, using both pre-trained Word2Vec embeddings and dynamically trained embedding layers within a Multi-Layer Perceptron (MLP). The study evaluates these approaches on text corpora, including *Le Comte de Monte-Cristo* by Alexandre Dumas, with results measured through perplexity scores and generated text quality. Both methods show similar performance, highlighting flexibility in embedding strategies, although the lack of a larger corpus limits definitive conclusions. The source code is available at https://github.com/Morpheus5828/StaticTokenDiv.

Mots clés— Neural network, word embedding

1. INTRODUCTION

In the previous report on the creation of word embedding using Word2Vec algorithm, a simple observation was realized, Markov model can't modelise some phenomenon ahead of model history. If we take a sentence in train corpus, for example : *John is driving around Eiffel tower*, test corpus contains this sample : *John is driving around ...*, a *n-gram* will return None because it doesn't see this example instead of a neural network model which says that *around* and *approximately* has almost same embeddings so next word could be *Eiffel tower*. Neural network classifier predict next word w_t knowing prefix $w_1, ..., w_{t-1}$. It compute probability distribution $P(w_t|w_1, ..., w_{t-1})$ for every words in vocabulary V.

2. MULTI-LAYER PERCEPTRON (MLP)

2.1. Formulation

In this section, a presentation will be given about how to create a multi-layer perceptron containing only dense layers. Let's discuss the model architecture. The first layer takes as input a tensor in $\mathbb{R}^{b,p}$, where b is the batch size and p is the product of k (the number of context words) and the embedding size. This layer outputs a tensor of size h_1. The second layer takes the output of the first layer (of size h_1) and transforms it into an output of size h_2. Finally, the third layer takes the output of the second layer (of size h_2) and produces an output of size equal to the vocabulary size, suitable for the classification task. For the activation function, the ReLU (Rectified Linear Unit) function is applied after each dense layer, except for the last one. The final layer uses a softmax activation function, as it returns probabilities corresponding to the different classes in the vocabulary. Indead, $y = softmax(y')$ can transform an y' vector which contains real value to a probaility vector y. Each components of y_i and y is compute as :

$$y_i = \frac{exp(y'_i)}{\sum exp(y'_j)_{j=1}^{|V|}} \tag{1}$$

The exp function will return positive value of $y'i$ and then normalize if by summing each y'_j values, it will promize than y_i is a probability vector because $\sum y_i{}_{i=1}^{|V|} = 1$.

Author can see code in the learning folder in code repository presented before.

Then, after presented model architecture, let's talk about the loss function which update W matrices during each iteration. For the reason, model will be a classification task, loss function will be Cross entropy H. Let's defince two probabilites set r and y on the same fundamental set $\Omega = \{1, ..., N\}$, $H(r, y)$ is computing like this :

$$-\sum_{i=1}^{N} P_r(i) log_2(P_y(i)) \tag{2}$$

If we consider r as the reference distribution and y as the model's predicted distribution, then $h(r, y)$ measures the error introduced when using y instead of r. In this case, r is a one-hot encoded distribution.

Given a vocabulary V where each word is indexed between 1 and V, the one-hot encoding of a word with index i is a vector of dimension $|V|$ where all components are 0, except for the i-th component, which is set to 1. When r is a one-hot encoded distribution, all terms in the computation are eliminated except for one. In this specific case, the cross-entropy is computed as follows :

$$H(r, c) = -log_2(y_i) \tag{3}$$

where i is target word index and y_i is the i componant of the last model layer ouput which associate a probability to a

word that user want to predict.

In the dataset, the text is composed of text without filter T, a word sequence divided by a sentence, and an embedding matrix EE generated using the Word2Vec algorithm. Before giving this dataset to the classifier, words must be transformed to their embedding by simple access to matrix E.

The classifier has to account for words unknown by vocabulary in the corpus (OOV for out of vocabulary). In fact, many words T don't appear in the vocabulary. To resolve this problem, words with an occurrence value less than λ will be replaced by a special token, for example : <unk>.

During the learning process, the classifier learns to predict these kinds of words as the special token present in the vocabulary. During prediction, if a word mm doesn't exist in the vocabulary, it will be replaced by <unk>. It is assumed that <unk> has an embedding accessible by the classifier during the learning process. To resolve this issue, the same method will be used : replace all words with an occurrence less than λ with <unk>, allowing the computation of an embedding for the special token <unk>.

2.2. Evaluation

In practice, evaluating a natural language processing model is a complex task. To achieve this, a test corpus is provided to two different models, and the model that assigns higher probabilities is deemed superior to the other. To evaluate model created using setting explain before, a solution is to compute model perplexity instead of probability by following this equation :

$$PP(T) = P(T)^{-\frac{1}{|T|}} \tag{4}$$

Where T is a test corpus. Perplexity and probability increase by opposition; when this one increases, the perplexity value decreases. This can be interpreted as the number of words the model can choose. This one is linked with cross-entropy loss.

2.3. Experimental Result

After implementing the model using Python programming language and PyTorch library, an experimental of this model has been computed on a corpus text from A.Dumas author called *Le Compte de Mont Cristo*. We use these parametters at the begining : *batch size* : 128, k : 2, the learning rate η : 0.001, optimizer : Adam and loss : Cross entropy. A demo code is available in demo folder at the Github link presented before to try it.

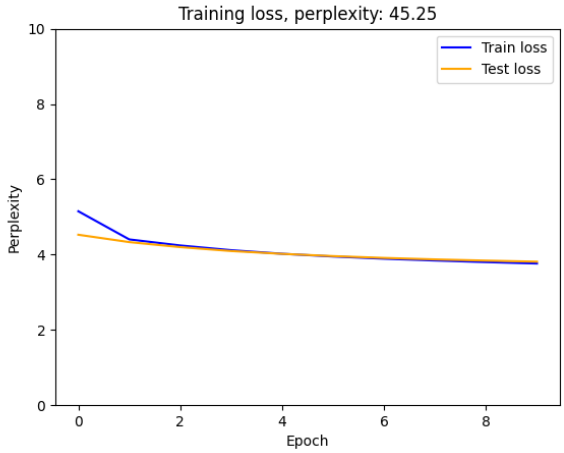

Fig. 1. Perplexity and loss values are better on *Le comte de Monte Cristo.train.unk5.tok* corpus. Vocab size is 224538.

and then text generation (the beginning of these sentences are *le poisson dans l eau*) :

Theta value	French generated sentence
0.1	le poisson dans l'eau de la main de
0.5	le poisson dans l'eau de la mer de
1	le poisson dans l'eau de sa première vieil
1.5	le poisson dans l'eau en malheur cette calèche
2	le poisson dans l'eau rospoli du secondes regardant

Table 1. Sentence generated on Le comte de Monte Cristo.train.unk5.tok corpus using Word2Vec embedding.

With theta equal to 0.5, next word is *mer* that is in the same context of *eau*, but for all of them sentence doesnt' have really sense. Another experiment was also realized by changing the corpus text. In fact, a fusion file which contains the fourth A. Dumas text has been done, then give this file to embedding file generator called Word2Vec (from project 2). With the same model parameter :

Theta value	French generated sentence
0.1	le poisson dans l'eau et de la vallière
0.5	le poisson dans l'eau et maintenant il y
1	le poisson dans l'eau par les épouvantés sous
1.5	le poisson dans l'eau brodée : la vive
2	le poisson dans l'eau mors opération legs lèse

Table 2. Sentence generated on fusion corpus using Word2Vec embedding.

Same thing like in previous table, with m_{i+1} and m_{i+2} for theta equal to 0.1 and 0.5, if reader has enough imagination, sentence can have sense but for the theta more than 1, it doesn't have any sence.

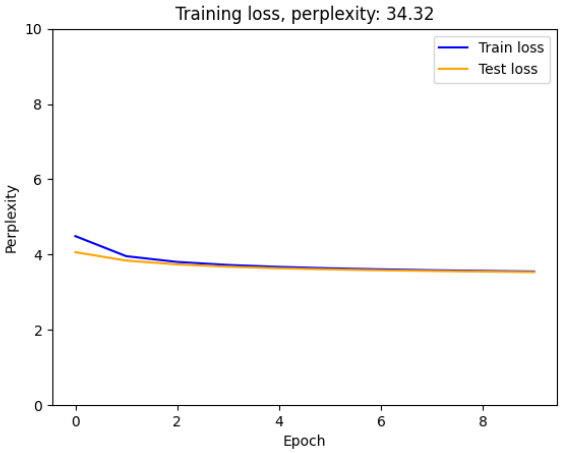

Fig. 2. Perplexity and loss values are better with fusionning corpus. Vocab size is 2410384.

3. TO GO FURTHER

3.1. Embedding layer

During the previous section, an embedding file was generated using our Word2Vec algorithm. But in fact, a replacement of this pre-workout was realized by adding a pytorch embedding layer in the model before dense layers. Let's try using fusion corpus.

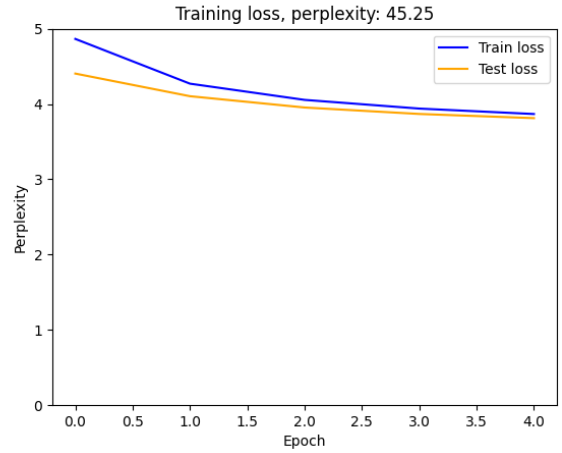

Fig. 3. Training loss of neural network model using embedding layer

As represented in Fig. 3, loss doesn't seems better than before on the previous model. Perplexity value increase than 10 % and text generation in table 3 doesn't seems better.

Theta value	French generated sentence
0.1	le poisson dans l'eau et de la reine
0.5	le poisson dans l'eau et à la porte
1	le poisson dans l'eau brillait m à ses
1.5	le poisson dans l'eau dent comme ce lait
2	le poisson dans l'eau autre portière me cas

Table 3. Sentence generated on fusion corpus using embedding pytorch layer.

3.2. Word2Vec VS Embedding layer

In comparing the use of pre-trained Word2Vec embeddings with the addition of an embedding layer directly trained within the neural network, the results revealed similar perplexity scores and generated sentence quality. While the two approaches yield comparable performance, it is challenging to conclude which method is superior definitively.

The pre-trained Word2Vec embeddings provide the advantage of being generated separately, leveraging extensive computational resources and possibly larger corpora, which can result in robust word representations. On the other hand, the embedding layer integrated into the neural network offers the flexibility of adapting embeddings during model training, potentially aligning them more closely with the specific task.

The similarity in outcomes suggests that both methods are viable for this application. However, lacking a larger and more diverse corpus limits the evaluation and prevents a more nuanced comparison. Future experiments with larger datasets and additional metrics could provide further insights into the advantages and trade-offs between these two embedding strategies.

3.3.

4. CONCLUSION

In conclusion, this report presented the implementation and evaluation of a neural network-based approach for word embedding and language modeling. The proposed architecture, leveraging a Multi-Layer Perceptron (MLP) and embedding techniques, demonstrated the ability to predict words and generate sentences in a given context effectively. The experimental results highlighted the impact of corpus size and quality on the model's perplexity and generated text coherence. [1]

By comparing pre-trained Word2Vec embeddings with dynamically trained PyTorch embedding layers, we observed similar performance metrics, suggesting flexibility in embedding initialization strategies depending on computational resources and dataset availability. Despite the observed limitations in generated sentence quality, the results underscore the potential of neural networks in natural language processing tasks.

Future work should focus on expanding the corpus to improve perplexity scores and exploring more advanced models such as Recurrent Neural Networks (RNNs) or Transformers to enhance text generation capabilities. These improvements could pave the way for more sophisticated applications in natural language understanding and generation.

5. REFERENCES

[1] James H. Martin Daniel Jurafsky, "Speech and language processing," 2024.